**Propósito**

Singleton es un patrón de diseño creacional que nos permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.

**Problema**

El patrón Singleton resuelve dos problemas al mismo tiempo, vulnerando el *Principio de responsabilidad única*:

-Garantizar que una clase tenga una única instancia. ¿Por qué querría alguien controlar cuántas instancias tiene una clase? El motivo más habitual es controlar el acceso a algún recurso compartido, por ejemplo, una base de datos o un archivo.

-Proporcionar un punto de acceso global a dicha instancia. ¿Recuerdas esas variables globales que utilizaste (bueno, sí, fui yo) para almacenar objetos esenciales? Aunque son muy útiles, también son poco seguras, ya que cualquier código podría sobrescribir el contenido de esas variables y descomponer la aplicación.

**Solución**

* Todas las implementaciones del patrón Singleton tienen estos dos pasos en común:
* Hacer privado el constructor por defecto para evitar que otros objetos utilicen el operador new con la clase Singleton.
* Crear un método de creación estático que actúe como constructor. Tras bambalinas, este método invoca al constructor privado para crear un objeto y lo guarda en un campo estático. Las siguientes llamadas a este método devuelven el objeto almacenado en caché.

**Aplicabilidad**

Utiliza el patrón Singleton cuando una clase de tu programa tan solo deba tener una instancia disponible para todos los clientes; por ejemplo, un único objeto de base de datos compartido por distintas partes del programa.

 El patrón Singleton deshabilita el resto de las maneras de crear objetos de una clase, excepto el método especial de creación. Este método crea un nuevo objeto, o bien devuelve uno existente si ya ha sido creado.

Utiliza el patrón Singleton cuando necesites un control más estricto de las variables globales.

Al contrario que las variables globales, el patrón Singleton garantiza que haya una única instancia de una clase. A excepción de la propia clase Singleton, nada puede sustituir la instancia en caché.

**Pros**

* Puedes tener la certeza de que una clase tiene una única instancia.
* Obtienes un punto de acceso global a dicha instancia.
* El objeto Singleton solo se inicializa cuando se requiere por primera vez.

## Contras

* Vulnera el *Principio de responsabilidad única*. El patrón resuelve dos problemas al mismo tiempo.
* El patrón Singleton puede enmascarar un mal diseño, por ejemplo, cuando los componentes del programa saben demasiado los unos sobre los otros.
* El patrón requiere de un tratamiento especial en un entorno con múltiples hilos de ejecución, para que varios hilos no creen un objeto Singleton varias veces.
* Puede resultar complicado realizar la prueba unitaria del código cliente del Singleton porque muchos *frameworks* de prueba dependen de la herencia a la hora de crear objetos simulados (mock objects). Debido a que la clase Singleton es privada y en la mayoría de los lenguajes resulta imposible sobrescribir métodos estáticos, tendrás que pensar en una manera original de simular el Singleton. O, simplemente, no escribas las pruebas. O no utilices el patrón Singleton.